The MVC Design Pattern & the Struts Framework

Separating Model (Data Access), Controller & View

- Development “Best practice”
- Known well before web items
  - Smalltalk pioneered
- Model: Access to Underlying Databases and Info Sources
- Controller: Control Flow of Web App
- View: Look-and-Feel

MVC originated as Model 2 in Web developers’ community

- Model 1: business logic is attached to jsp
  - Similar to students.jsp of JDBC lectures
  - Similar to project’s Phase 1
- Model 2: data access and control flow decisions in beans
The Larger Issue: Specification and Modularization

- Frictions in specification
- Inefficiencies in Large Project Management

The Process and the Frictions

<table>
<thead>
<tr>
<th>Business Process Owner (Client)</th>
<th>Analysis/Specification Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMUNICATION</td>
<td>business process and specification of Web application</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chief Architect/Technical Project Leader</th>
<th>Development Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMUNICATION</td>
<td>technical specification and development</td>
</tr>
</tbody>
</table>

| Developer | Problem is even worse in evolution phase when business logic is hidden in thousands of code lines |

The Problem: Communication
Magnitude of the Problem Increases

- Business processes implemented as Web apps
- More complex business processes become Web applications
  - Beyond basic e-commerce and e-publishing
  - Drug discovery/development scheduling
  - Multi-step, multi-participant application selection and approval processes (loans, land development, hiring, ...)
- Collaboration by remote teams
  - Pressure on CIOs to off-shore, yet 60% of CIOs declare that off-shoring experience "has been hell" [Gartner]
  - Offshoring can reduce cost to 30%, but in practice only to 65% [Gartner]

From Requirements to Web Application Implementations
Step 1: Demos and Site Diagrams for the Customer

Interview with Committee Chair

- Overview of status of review process. Present numbers per category
  - Foreign/Domestic (citizenship)
  - Status: admitted, star admissible, admissible, rejected, reviewed, under review (status)
  - Degree applied for (degree)
- See the list of applicants for each category
  - Along with number of reviews per applicant
  - Be able to search by country or research area preference
Applicant Data split among pages for
- Profile
  - Foreign/Domestic, Country, Area, Multiple Degrees...
- References
- CV & Statement
- Reviews
- One click navigation

Q: How are applicants identified? Student id

Decide status
- Filter applicants by (combination of any of)
  status, citizenship, degree, area of preference
- Pick applicant and change status to any of
  admitted, admissible, star admissible, reject and withdrawn

Specifications of Web Applications as State Transition Diagrams
public enum DegreeType { PhD, MSc }

public enum CitizenshipType { Domestic, Foreign, Total }

public enum StatusID { Admitted, Admittable }

Map<DegreeType, Map<CitizenshipType, Map>StatusID, Integer>> summary;

### Struts: black-box framework that implements MVC

- **Framework**: reusable "partial" application
  - Struts' ActionServlet provides high level control of workflow
- **You provide beans and files to customize framework into your application**
  - Jsp's provide HTML presentation
  - ActionForm beans "collect" form data
  - Action beans provide details of flow
  - Struts-config.xml declares beans and jsp's

### How to Develop a Struts Applications – 10 miles high

- **Pass high-level control** to ActionServlet
  - By appropriate URL mapping in web.xml
- **Design "master plan"** in struts-config.xml
- **Develop ActionForm bean(s)**
- **Develop Action bean(s)**
- **Develop model beans**
- **Develop html and jsp pages**
The Art of Balancing How Many Actions & JSP’s to Use

- Consider the "logon" application of the textbook
- We could have one jsp for each kind of login error
- However, we will see technologies that help consolidate within a few jsp’s
  - Form validation features
  - Logic tag library
- Deciding the number of actions and jsp’s is an art of design – not a science
  - Examples, practice, then more practice...

WorkFlow

```
WorkFlow
```

```
/pages/Welcome.jsp
```

```
&lt;%= taglib url="/tags/struts-bean" prefix="bean" %
&lt;%= taglib url="/tags/struts-html" prefix="html" %
&lt;%= taglib url="/tags/struts-logic" prefix="logic" %
&lt;/HTML&gt;

&lt;TITLE>Welcome</TITLE&gt;
&lt;html:base/&gt;
&lt;/HEAD&gt;

&lt;h3&gt;Welcome &lt;/h3&gt;

&lt;h3&gt;Welcome World!</h3&gt;

&lt;ul&gt;
  &lt;li&gt;&lt;html:link forward="/logon"&gt;Sign in&lt;/html:link&gt;&lt;/li&gt;
  &lt;li&gt;&lt;html:link forward="/logoff"&gt;Sign out&lt;/html:link&gt;&lt;/li&gt;
&lt;/ul&gt;

&lt;img src='struts-power.gif' alt='Powered by Struts'/&gt;
```
Taglib declarations required in web.xml

<!-- Struts Tag Library Descriptors -->
<taglib>
  <taglib-uri>/tags/struts-bean</taglib-uri>
  <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>
<taglib>
  <taglib-uri>/tags/struts-html</taglib-uri>
  <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
</taglib>
<taglib>
  <taglib-uri>/tags/struts-logic</taglib-uri>
  <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
</taglib>

Forward definitions in struts-config.xml

<!-- ========= Global Forward Definitions -->
<global-forwards>
  <forward name="logoff" path="/Logoff.do"/>
  <forward name="logon" path="/Logon.do"/>
  <forward name="welcome" path="/Welcome.do"/>
</global-forwards>

logon.jsp

<%@ taglib uri="/tags/struts-html" prefix="html" %>
<HTML>
  <HEAD>
    <TITLE>Sign in, Please!</TITLE>
  </HEAD>
  <BODY>
    <html:errors/>
    <html:form action="/LogonSubmit" focus="username">
      <TABLE border="0" width="100%">
        <TR>
          <TH align="right">Username:</TH>
          <TD align="left"><html:text property="username"></html:text></TD>
        </TR>
        <TR>
          <TH align="right">Password:</TH>
          <TD align="left"><html:password property="password"></html:password></TD>
        </TR>
        <TR>
          <TD align="right"><html:submit/></TD>
          <TD align="left"><html:reset/></TD>
        </TR>
      </TABLE>
    </html:form>
  </BODY>
</HTML>
Associating the ActionForm bean with the Form

```xml
<action
    path="/LogonSubmit"
    type="app.LogonAction"
    name="logonForm"
    scope="request"
    validate="true"
    input="/pages/Logon.jsp">
    <forward
        name="success"
        path="/pages/Welcome.jsp"/>
</action>
```

Form Beans Also Provide Values

```xml
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<HTML>
<HEAD>
<TITLE>Sign in, Please!</TITLE>
</HEAD>
<BODY>
<html:errors/>
<html:form action="/LogonSubmit" focus="username">
<TABLE border="0" width="100%">
<TR>
<TD align="right">Username:</TD>
<TD align="left"><html:text property="username"></html:text></TD>
</TR>
<TR>
<TD align="right">Password:</TD>
<TD align="left"><html:password property="password"></html:password></TD>
</TR>
<TR>
<TD align="right"><html:submit/></TD>
<TD align="left"><html:reset/></TD>
</TR>
</TABLE>
```

The corresponding ActionForm bean will also be used to populate html form.

Typical code of a LogonForm (may use a tool to write)

```java
public final class LogonForm extends ActionForm {
    private String password = null;
    private String username = null;

    public String getPassword() { return (this.password); }
    public void setPassword(String password) {
        this.password = password;
    }

    public String getUsername() { return (this.username); }
    public void setUsername(String username) {
        this.username = username;
    }

    public void reset(ActionMapping mapping, HttpServletRequest request) {
        setPassword(null);
        setUsername(null);
    }
```
Validation

```java
public ActionErrors validate(ActionMapping mapping,
HttpServletRequest request) {
    ActionErrors errors = new ActionErrors();
    if ((username == null) || (username.length() < 1))
        errors.add("username",
                  new ActionError("error.username.required"));
    if ((password == null) || (password.length() < 1))
        errors.add("password",
                  new ActionError("error.password.required"));
    return errors;
}
```

Resource File & Internationalization

```javascript
welcome.title=Struts Logon Example Application
welcome.heading=Welcome!
welcome.message=To get started on your own application, copy the struts-blank.war to a new W
equivalent), and let your container auto-deploy the application. Edit the skeleton configuration f
application.properties file with this message in the <B>/WEB-INF/src/java/resources</B> folder.
errors.footer=<HR></UL><HR>
errors.header=<H3><FONT color="red">Validation Error</FONT></H3>You must correct the
error.username.required=<LI>Username is required</LI>
error.password.required=<LI>Password is required</LI>
error.logon.invalid=<LI>Username and password provided not found in user directory. Password
error.logon.connect=<LI>Could not connect to user directory.</LI>
```

Action Bean Logon

```java
package app;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionServlet;

public final class LogonAction extends Action {
```

9
/**
 * Validate credentials with business tier.
 * @param username The username credential
 * @param password The password credential
 * @returns true if credentials can be validated
 * @exception UserDirectoryException if cannot access directory
 */
public boolean isUserLogon(String username,
String password) throws UserDirectoryException {
    return (UserDirectory.getInstance().isValidPassword(username,password));
// return true;
}

public ActionForward perform(ActionMapping mapping,
ActionForm form,
HttpServletRequest request,
HttpServletResponse response)
throws IOException, ServletException {
    // Obtain username and password from web tier
    String username = ((LogonForm) form).getUsername();
    String password = ((LogonForm) form).getPassword();
    // Validate credentials with business tier
    boolean validated = false;
    try {
        validated = isUserLogon(username,password);
    }
    catch (UserDirectoryException ude) {
        // couldn't connect to user directory
        ActionErrors errors = new ActionErrors();
        errors.add(ActionErrors.GLOBAL_ERROR,
        new ActionError("error.logon.connect");
        saveErrors(request,errors);
        // return to input page
        return (new ActionForward(mapping.getInput()));
    }
    if (!validated) {
        // credentials don't match
        ActionErrors errors = new ActionErrors();
        errors.add(ActionErrors.GLOBAL_ERROR,
        new ActionError("error.logon.invalid");
        saveErrors(request,errors);
        // return to input page
        return (new ActionForward(mapping.getInput()));
    }
    // Save our logged-in user in the session,
    // because we use it again later.
    HttpSession session = request.getSession();
    session.setAttribute("user", form);
    // Log this event, if appropriate
    if (servlet.getDebug() == Constants.DEBUG) {
        StringBuffer message =
        new StringBuffer("LogonAction: User ");
        message.append(username);
        message.append(" logged on in session ");
        message.append(session.getId());
        servlet.log(message.toString());
    }
    // Return success
    return (mapping.findForward("success");
}
From Customer-Oriented Specifications to Architectures

public enum DegreeType {PhD, MSc}
public enum CitizenshipType {Domestic, Foreign, Total}
public enum StatusID {Admitted, Admittable}

Map<DegreeType, Map<CitizenshipType, Map>StatusID, Integer>> summary;